FactSet Exchange Data Feed Snapshot Service

Programmer's Manual and Reference
Version 1.2N

Table of Contents

Notice

This manual contains confidential information of FactSet Research Systems Inc. or its affiliates ("FactSet"). All proprietary rights, including intellectual property rights, in the Licensed Materials will remain property of FactSet or its Suppliers, as applicable. The information in this document is subject to change without notice and does not represent a commitment on the part of FactSet. FactSet assumes no responsibility for any errors that may appear in this document.

FactSet Consulting Services

North America - FactSet Research Systems Inc.	
United States and Canada	+1.877.FACTSET
Europe – FactSet Limited	
United Kingdom	0800.169.5954
Belgium	800.94108
France	0800.484.414
Germany	0800.200.0320
Ireland, Republic of	1800.409.937
Italy	800.510.858
Netherlands	0800.228.8024
Norway	800.30365
Spain	900.811.921
Sweden	0200.110.263
Switzerland	0800.881.720
European and Middle Eastern countries not listed above	+44.(0)20.7374.4445
Pacific Rim- FactSet Pacific Inc.	
Japan Consulting Services (Japan and Korea)	0120.779.465 (Within Japan) +81.3.6268.5200 (Outside Japan)
Hong Kong Consulting (Hong Kong, China, India, Malaysia, Singapore, Sri Lanka, and Taiwan)	+852.2251.1833
Sydney Consulting Services	1800.33.28.33 (Within Australia) +61.2.8223.0400 (Outside Australia)
E-mail Support	
	support@factset.com

Document Organization and Audience

This document describes how to use the FactSet Exchange DataFeed Snapshot Service. You should be familiar with the XML language, the HTTP protocol, and Web Services. This document will describe the syntax needed for proper request formatting as well as the rules for processing responses. In addition, complete code examples are included, which further illustrate the use of this service.

- Chapter 1 is an introduction to the Exchange Snapshot Service with general concepts and terminology.
- Chapter 2 explains how to request data using the HTTP GET and POST methods.
- Chapter 3 details the XML and CSV responses, as well as the types of errors that can be received.
- Chapter 4 describes the use of portfolios with this service.
- Chapter 5 describes the Data Link use case.
- The Appendix supplements this document by providing additional code examples and a list of recent changes to the document.

Document Conventions

This document uses the following conventions:

- Code examples use a courier 10 font - int i = 0;
- Items of importance will be in boxes of following type:

❖ ***Important notations will be in this type of box.***

Trademarks

Acknowledgements

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org).

Chapter 1 Introduction to General Concepts

1.1 FactSet Exchange DataFeed Snapshot Service

The FactSet Exchange DataFeed Snapshot Service provides synchronous access to data via the standard HTTPS protocol (see *Figure 1*). Data is returned as a well-described XML document or CSV file. Responses for multiple data elements can be forced into a table-like structure, which further simplifies application development.

This service uses the FactSet DataFeed API and thus the same data model. The *FactSet Data Service Specification* describes the data fields, types, and possible values for each type of record.

Clients of the Exchange Snapshot Service will be given a secure URL, username, and password. The user information is used in each request to authenticate and permission each data item.

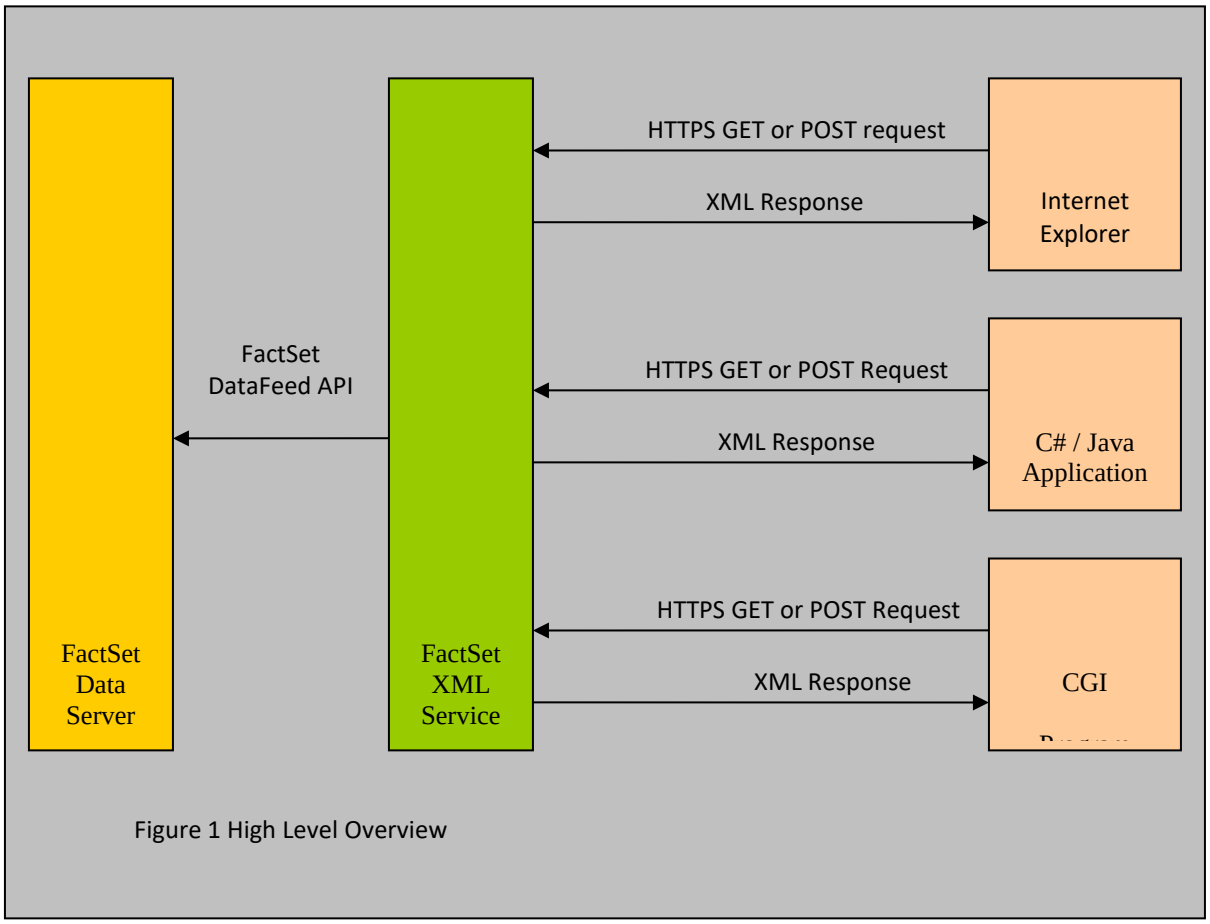

Figure 1 High Level Overview

1.2 Terminology

The following terminology is used throughout this documentation:

Terminology	Meaning
API	Application Programming Interface - a set of defined interfaces that applications use to extract information from the FactSet Data Server.
SDK	Software Development Kit - a collection of libraries, include files, documentation, and sample code that makes up this toolkit.
XML	eXtensible Markup Language - a defined standard for exchanging information. The information contains markup tags used to describe the data values.
CSV	Comma Separated Value - a table-like structure that separates fields via a delimiter (usually a comma ","). The first header row describes each of the column names.
TCP/IP	Transport Control Protocol over Internet Protocol - one of the protocols that this service uses to communicate with the FactSet Data Server.
FactSet Data Server	A server that provides permissioned access to FactSet data.
FDS	Multiple meanings. FDS is the ticker symbol for FactSet Research Systems Inc. It is also the C++ namespace that encapsulates the FactSet DataFeed API. Finally, it may stand for the FactSet Data Server. The meaning is defined by its context.
Service	A data source or supplier identified by a string name.
FDS1	FactSet's Streaming Production Data Service. For a complete description of the data fields, types, and possible values see the FactSet Data Service Manual.
FID	Field Identifier - an integer identifier that describes the encoding and business meaning of a field value.
Opaque Data	Data without a defined interpretation, which is simply a pointer to data and a size.
Field/Value Pairs	A self-describing message format used in API responses. Each pair contains a FID and some opaque data. The FID defines the type and meaning of the data.
HTTP	Hypertext Transfer Protocol - the protocol that governs how information is exchanged over a network.
HTTPS	Secure Hypertext Transfer Protocol - the protocol that uses SSL to securely communicate HTTP messages.
SSL	Secure Sockets Layer - a protocol used to encrypt data over an insecure medium.
URL	Uniform Resource Locator - used to identify a resource when using HTTP on the Internet.

1.3 Exchange Snapshot Service Core Functionality and Benefits

The Exchange Snapshot Service provides the following features to applications:

- Synchronous access using standard HTTP methods.
- Consistent data model that matches the underlying C++ DataFeed API.
- Stateless requests (No sticky cookies).
- Platform independence.
- Zero-based installation.
- Batched requests allowing multiple items to be requested at the same time.
- Permissioned data access.
- Seamless failover.
- Production and beta environments.
- Easy access to FactSet's web-based products through FactSet OnDemand.

1.3.1 Synchronous Access

Synchronous data access simplifies application programming and makes it straightforward to exploit the HTTP protocol.

1.3.2 Consistent Data Model

The Exchange Snapshot Service uses the same well-defined data model that is defined in the *FactSet Data Service Specification*. This makes it easier to mix API and XML applications. In addition, extensions made to the FactSet data model will automatically be available via Exchange Snapshot.

1.3.3 Stateless Request/Response

The Exchange Snapshot Service is a stateless request/response service. The service does not make any use of HTTP cookies, and thus allows requests to be load-balanced over many servers. This in turn, improves both throughput and response time.

1.3.4 Platform Independence

The Exchange Snapshot Service can be used without the any FactSet-supplied software. Applications can access data from any system that supports TCP/IP.

1.3.4 Zero-Based Installation

Proprietary software is not needed to access this service. Any standard library or application that can issue HTTP queries can request data via the Exchange Snapshot Service.

1.3.5 Batched Requests

The Exchange Snapshot Service allows applications to request multiple items at a time. The records can be filtered by the fields of interest, and the entire response can be forced into a matrix or table. This simplifies client-side development and improves performance. Applications are encouraged to take advantage of this feature.

1.3.6 Permissioned Access

The Exchange Snapshot Service handles all exchange permissioning on behalf of the user defined in each request. Therefore, multi-user proxy-type applications can easily be created, provided that the users are already using FactSet applications.

1.3.7 Seamless Failover

If a FactSet Data Server or an entire data center goes down, requests will be routed to an available Data Server using state-of-the-art load balancers. You do not have to change your software, as this failover will be automatic.

1.3.8 Production and Beta Environments

FactSet provides two independent environments for programmatic access. One is for production and the other for beta. The following table shows the URLs needed for each system.

Type	Host Name	Service	Example URL[1]
Production	datadirect.factset.com	DFSnapshot	https://datadirect.factset.com/services/DFSnapshot
Beta	datadirect-beta.factset.com	DFSnapshot	https://datadirect-beta.factset.com/services/DFSnapshot

- **You will start on the beta network until your software is ready for production. In addition, the FDS_C data service is used until exchange agreements have been submitted. Therefore, this document will use the beta URL and FDS_C data service for all examples and sample codes. Once you move to a production environment, you will need to update both the URL and data service.**
- **You should contact FactSet to ensure your production permissions meet your requirements as permissions in the beta environment may differ from the production environment.**
- **The FDS_C data service is yesterday's market open replayed throughout the day.**

1.3.9 Easy Access through FactSet OnDemand

Since the Exchange Snapshot Service is part of FactSet OnDemand, it provides easy access to FactSet's web-based products. All services are authenticated by a central source, so the same username and password will work for all services. In addition, naming conventions have been standardized to simplify the consumption of multiple services.

FactSet OnDemand uses HTTP Basic Authentication. This scheme requires the following HTTP header be added to the request:

Authorization: Basic {Base-64 Encoding of "user:password"}

The exact header will be provided by FactSet along with the FactSet OnDemand username and password.

- **While the standard scheme is HTTP Basic Authentication, there are stronger types of authentication provided by FactSet OnDemand if preferred.**

1.3.10 Compressed Responses

To ensure the best response time please ensure that you have the ability to accept compressed responses. FactSet will not accept requests that do not have the ability to accept a compressed response for bandwidth savings and performance improvement.

Additional information on this requirement and the Industry standard can be found here.

http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.3

1.3.11 Certificates and Certificate Chains

Instead of hardcoding reliance on any particular certificate or advertised certificate chain into applications FactSet expects clients to rely on Public Key Infrastructure verification and validity of the certificates:

- FactSet's Certificates will change over time as they are renewed and the complexity of the algorithms employed increases (i.e. SHA-2 rather than SHA-1 signatures). This is the constant evolution of security as old security algorithms are retired and new

[1] The example URLs demonstrate the different services assigned to production and beta. The URLs given above may not be valid. For example, the query string is missing in the above example.

security algorithms are included. Often these certificates get updated on a rolling multi-year basis. Validating a certificate dynamically during TLS connect must be incorporated by the client as a necessary practice.

- FactSet's Certificate chains, including intermediate Certificate Authorities, may change over time, and it is important that clients dynamically validate FactSet's certs against a modern CA trusted root certificate store. FactSet's current root certificate is Thawte Primary Root CA.

1.3.12 Security Protocols

Clients should not hardcode dependencies on any specific security protocol as FactSet is continuously reviewing security policies and reserves the right to disable support for older security protocols with short notice[2]. The current supported protocols are TLSv1.1 and TLSv1.2 but at a future date, these may be replaced with future versions. Clients should make sure that their software can handle ever changing Security Protocols.

[2] As of 29-Jul-2017 support for security protocol TLSv1.0 is disabled and requests using this TLS version will fail.

Chapter 2 HTTP GET and POST Requests

2.1 HTTP GET

When using an HTTP GET request, the search criteria is sent via the query string in the URL.

Example

This request returns the BID_1, ASK_1, and LAST_1 fields for the symbol FDS-USA on the FDS_C service.

https://datadirect-beta.factset.com/services/DFSnapshot?serv=FDS_C&req_id=99&format=xml&ids=FDS-USA&fields=BID_1,ASK_1,LAST_1

with the response:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
<Response Service="FDS_C">
  <Request id="99">
    <RequestedSymbols>FDS-USA,IBM-USA,TWX-USA</RequestedSymbols>
    <RequestedFields>ASK_1,BID_1,LAST_1</RequestedFields>
    <Host>dfstagea01[3978]</Host>
  </Request>
  <Error code="0" description="" />
  <Records>
    <Record req_sym="FDS-USA" key="FDS-USA" stale="">
      <Fields>
        <Field id="200" name="ASK_1" value="43.75" />
        <Field id="100" name="BID_1" value="43.73" />
        <Field id="300" name="LAST_1" value="43.73" />
      </Fields>
    </Record>
  </Records>
</Response>
```

If no fields are specified all available fields for the requested security are returned. The below request returns all available fields for the symbol FDS-USA on the FDS_C service.

https://datadirect-beta.factset.com/services/DFSnapshot?serv=FDS_C&req_id=99&format=xml&ids=FDS-USA

- **HTTP GET requests will meet the needs of most applications. However, some systems limit the size of a URL. For example, Internet Explorer has a maximum URL length of 2,048 characters. Since the size of the URL depends on the query string, this limitation can create some application complexity. As an alternative, HTTP POST requests, which are not burdened by this limit, can be used.**

2.2 Query Variables

Query Variable	Description
Req_Id	Request Identification String. Can be used by the application to keep track of requests. The id is not used by this service; however, it is included in the XML response.
Format	Response Format. Possible options: xml – XML Response csv – CSV Response json – JSON Response html – HMTL Response (used for testing/debugging)
Serv	Data Service - the service that handles the request for the specified symbols. Current available data services: *"FDS1" – FactSet's Streaming Production Data Service* *"FDS_C" – FactSet's Canned Data Service. Recorded data is replayed, used for testing.* *"FDS_FUND" – FactSet's Fundamental Data Service. Used for End of Day data.*
Ids	Requested symbols or securities. This is a comma-separated list with a maximum of 500. Each symbol can be a FactSet exchange symbol, CUSIP, or SEDOL.
Fields	Requested fields. This is also a comma-separated list (no spaces). If not specified, all available fields are returned.

2.3 Example Codes for HTTP GET

All of the example codes request the ASK_1, BID_1, and LAST_1 fields for the symbols, FDS-USA, IBM-USA, and TWX-USA. Two programming environments are illustrated (C#/.NET and Java). Each programming language has three sections. The first section makes the HTTP GET request, the second parses the XML response, and the third displays the run-time output. The examples have been tested and can be used as-is provided that the HTTP Basic Authorization header is modified appropriately[3].

2.3.1 Using C# (.NET 2.0)

Issuing an HTTP GET request

```
using System.IO;
using System.IO.Compression;
using System.Net;
using System.Xml;

class MyXMLParser
{

[System.STAThread]
static void Main(string [] args)
{
  // setup the request
  HttpWebRequest req = (HttpWebRequest)WebRequest.Create(
            "https://datadirect-beta.factset.com/services/DFSnapshot?" +
            "serv=FDS_C&req_id=99&format=xml" +
            "&ids=FDS-USA,IBM-USA,TWX-USA&fields=ASK_1,BID_1,LAST_1"
  req.KeepAlive = false;
  req.Headers.Add("Accept-Encoding", "deflate, gzip");
  req.Headers.Add("Authorization", "Basic AaBbCcDdEeFfGgHhIi1234==");

  // make request and get the response
  HttpWebResponse rsp = (HttpWebResponse)req.GetResponse();
  Stream stream = rsp.GetResponseStream();

  // uncompress the response
  if (rsp.ContentEncoding.Equals("deflate"))
    stream = new DeflateStream(stream, CompressionMode.Decompress);
  else if (rsp.ContentEncoding.Equals("gzip"))
    stream = new GZipStream(stream, CompressionMode.Decompress);
  .
  .
```

[3] The username and password, as well the exact HTTP header to add to the request, will be supplied by FactSet.

Parsing the XML response

This code uses a C# XMLTextReader to move through the response nodes and print out the values[4]

```
 .
 .
 .
 XmlTextReader reader = new XmlTextReader(stream);
 reader.WhitespaceHandling = WhitespaceHandling.None;

 while (reader.Read())
 {
   if (reader.NodeType.Equals(XmlNodeType.Element))
   {
     if (reader.Name.Equals("Error"))
     {
       System.Console.WriteLine(reader.Name);
       printAttributes(reader);
       System.Console.WriteLine();
     }
     else if (reader.Name.Equals("Record"))
     {
       System.Console.WriteLine(reader.Name);
       printAttributes(reader);

       // Move to first Field element
       while (reader.Read() && !reader.Name.Equals("Field"))
       { }

       // Print out all field ids, names and values
       do
       {
         printAttributes(reader);
       } while (reader.Read() && reader.Name.Equals("Field"));
       System.Console.WriteLine();
     }
   }
 }
}

static void printAttributes(XmlTextReader reader)
{
  while (reader.MoveToNextAttribute())
    System.Console.Write(reader.Name + ": " + reader.Value + ", ");
  System.Console.WriteLine();
}

} // class MyXMLParser
```

[4] Applications may also use an XMLDocument to parse the response, see Appendix A.

Program output

```
Error
code: 0, description: ,

Record
req_sym: FDS-USA, key: FDS-USA, stale: ,
id: 200, name: ASK_1, value: 43.65,
id: 100, name: BID_1, value: 43.63,
id: 300, name: LAST_1, value: 43.65,

Record
req_sym: IBM-USA, key: IBM-USA, stale: ,
id: 200, name: ASK_1, value: 75.68,
id: 100, name: BID_1, value: 75.67,
id: 300, name: LAST_1, value: 75.67,

Record
req_sym: TWX-USA, key: TWX-USA, stale: ,
id: 200, name: ASK_1, value: 16.36,
id: 100, name: BID_1, value: 16.35,
id: 300, name: LAST_1, value: 16.36,
```

2.3.2 Using Java

Issuing an HTTP GET request

This example uses Java to make a web request and parse the response.

❖ **Since there is a java.net.ContentHandler and a org.xml.sax.ContentHandler, applications should avoid the following code:**
Import java.net.*;
Import org.xml.sax.*;
Import java.net.*;
Import org.xml.sax.*;

```
import java.io.*;
import java.util.zip.*;
import java.net.URL;
import javax.net.ssl.HttpsURLConnection;
import javax.xml.parsers.*;
import org.xml.sax.*;

public class MyXMLParser implements ContentHandler {

public static void main(String[] args)
{
  try {
    // setup the request
    URL url = new URL(
                                          "https://datadirect-beta.factset.com/services/DFSnapshot?" +
                                                "serv=FDS_C&req_id=99&format=xml" +
                                        "&ids=FDS-USA,IBM-USA,TWX-USA&fields=ASK_1,BID_1,LAST_1"
    HttpsURLConnection conn = (HttpsURLConnection)url.openConnection();
    conn.setDoOutput(true);
    conn.setRequestProperty("Accept-Encoding", "deflate, gzip");
    conn.setRequestProperty("Authorization",
                    "Basic AaBbCcDdEeFfGgHhIi1234==");

    // getInputStream will implicitly connect and get the response
    InputStream inputStream = conn.getInputStream();

                                                    // uncompress the response
    if (conn.getContentEncoding().equals("deflate"))
      inputStream = new InflaterInputStream(inputStream,
```

```
                                new Inflater(true));
    else if (conn.getContentEncoding().equals("gzip"))
      inputStream = new GZIPInputStream(inputStream);
    .
    .
    .
```

Parsing the XML response

The SAX parser, which comes with the Standard Edition of Java, uses event-handler callbacks for each element in the XML document[5]. Therefore, in order to print the record attributes for each field, the "req_sym", "key", and "stale" values must be saved in temporary variables.

```
    .
    .
    XMLReader reader =
       SAXParserFactory.newInstance().newSAXParser().getXMLReader();
    reader.setContentHandler(new MyXMLParser());
    reader.parse(new InputSource(inputStream));
  } catch (IOException e) {
    // Deal with IOException
  } catch (SAXException e) {
    // Deal with SAXException
  } catch (ParserConfigurationException e) {
    // Deal with ParserConfigurationException
  }
}

public void startElement(String uri, String localName, String qName,
                         Attributes atts)
{
    if (qName.equals("Error"))
      System.out.println("Error code: " + atts.getValue("code")
        + ", description: " + atts.getValue("description"));
    else if (qName.equals("Record")) {
      currentReqSym = atts.getValue("req_sym");
      currentKey    = atts.getValue("key");
      currentStale  = atts.getValue("stale");
    }
    else if (qName.equals("Field"))
      System.out.println("req_sym: " + currentReqSym
        + ", key: " + currentKey + ", stale: " + currentStale
        + ", id: " + atts.getValue("id")
        + ", name: " + atts.getValue("name")
        + " = " + atts.getValue("value"));
}

private String currentReqSym, currentKey, currentStale;

public void characters(char [] ch, int start, int length)         {}
public void endDocument()                                          {}
public void endElement(String uri, String localName, String qName) {}
public void endPrefixMapping(String prefix)                        {}
public void ignorableWhitespace(char[] ch, int start, int length)  {}
public void processingInstruction(String target, String data)      {}
public void setDocumentLocator(Locator locator)                    {}
public void skippedEntity(String name)                             {}
public void startDocument()                                        {}
public void startPrefixMapping(String prefix, String uri)          {}

} // class MyXMLParser
```

[5] Applications may also use an XPath and Document object to parse the response, see Appendix B.

Program output

```
Error code: 0, description:
req_sym: FDS-USA, key: FDS-USA, stale: , id: 200, name: ASK_1 = 43.85
req_sym: FDS-USA, key: FDS-USA, stale: , id: 100, name: BID_1 = 43.83
req_sym: FDS-USA, key: FDS-USA, stale: , id: 300, name: LAST_1 = 43.84
req_sym: IBM-USA, key: IBM-USA, stale: , id: 200, name: ASK_1 = 76.10
req_sym: IBM-USA, key: IBM-USA, stale: , id: 100, name: BID_1 = 76.08
req_sym: IBM-USA, key: IBM-USA, stale: , id: 300, name: LAST_1 = 76.09
req_sym: TWX-USA, key: TWX-USA, stale: , id: 200, name: ASK_1 = 16.33
req_sym: TWX-USA, key: TWX-USA, stale: , id: 100, name: BID_1 = 16.32
req_sym: TWX-USA, key: TWX-USA, stale: , id: 300, name: LAST_1 = 16.32
```

2.4 HTTP POST

A POST request can contain an HTTP body. The Exchange Snapshot Service expects a string identical to the query string (used in an HTTP GET request) to be embedded within the body of a POST request. This format is also used by HTML forms. For an example form, see section *2.4.1 Example HTML Form.*

All post requests should use the following URL:

```
https://datadirect-beta.factset.com/services/DFSnapshot
```

The POST body should contain name/value pairs separated by the "&" character. This format is identical to the query string used in HTTP GET requests. The variables allowed are also the same and can be found in section 2.2 Query Variables.

POST Bodies Examples

This example returns all available fields for the symbol FDS-USA on the FDS_C service.

```
serv=FDS_C&req_id=99&format=xml&ids=FDS-USA
```

This example returns the BID_1, ASK_1, and LAST_1 fields for the symbols, FDS-USA, IBM-USA, and TWX-USA, on the FDS_C service.

```
serv=FDS_C&req_id=99&format=xml&ids=FDS-USA,IBM-USA,TWX-USA& fields=BID_1,ASK_1,LAST_1
```

2.4.1 Example HTML Form

This HTML form can be used by any web browser to submit a request using HTTP POST.

```
<HTML>
<HEAD>
 <TITLE> FactSet DataFeed Snapshot Service </TITLE>
</HEAD>
<BODY>
 <form action="https://datadirect-beta.factset.com/services/DFSnapshot"
      method="POST">
  <H3>FactSet DataFeed Snapshot Service</H3>
  <p>Enter your search options: <BR>
   <table ID="Table1">
    <tr>
     <td> Request ID: </td>
     <td> <input type="text" name="req_id"> </td>
    </tr>
    <tr>
     <td> Format: </td>
     <td> <input type="text" name="format"> </td>
    </tr>
    <tr>
     <td> Service: </td>
     <td> <input type="text" name="serv"> </td>
    </tr>
    <tr>
```

```
        <td> Symbols: </td>
        <td> <input type="text" name="ids"> </td>
      </tr>
      <tr>
        <td> Fields: </td>
        <td> <input type="text" name="fields"> </td>
      </tr>
    </table>
    <BR>
    <input type="submit">
    <input type="reset">
  </p>
 </form>
</BODY>
</HTML>
```

2.5 Example Codes for HTTP POST

All of the example codes request the ASK_1, BID_1, and LAST_1 fields for the symbols, FDS-USA, IBM-USA, and TWX-USA. Two programming environments are illustrated (C#/.NET and Java). Each programming language has three sections. The first section makes the HTTP POST request, the second parses the XML response, and the third displays the run-time output6. The examples have been tested and can be used as-is provided that the HTTP Basic Authorization header is modified appropriately.

2.5.1 Using C# (.NET 2.0)

Issuing an HTTP POST request

```
using System.IO;
using System.IO.Compression;
using System.Net;
using System.Xml;
using System.Text;

class MyXMLParser
{
[System.STAThread]
static void Main(string [] args)
{
  HttpWebRequest req = (HttpWebRequest)WebRequest.Create(
                "https://datadirect-beta.factset.com/services/DFSnapshot");
  req.Method = "POST";
  req.KeepAlive = false;
  req.Headers.Add("Accept-Encoding", "deflate, gzip");
  req.Headers.Add("Authorization", "Basic AaBbCcDdEeFfGgHhIi1234==");
  string postData = "serv=FDS_C&req_id=99&format=xml" +
          "&ids=FDS-USA,IBM-USA,TWX-USA&fields=ASK_1,BID_1,LAST_1";
  byte[] byteArray = Encoding.ASCII.GetBytes(postData);
  req.ContentLength = byteArray.Length;

  // Fill in the Request Stream with the POST data
  Stream stream = req.GetRequestStream();
  stream.Write(byteArray, 0, byteArray.Length);
  stream.Close();

  // make request and get the response
  HttpWebResponse rsp = (HttpWebResponse)req.GetResponse();
  stream = rsp.GetResponseStream();

  // uncompress the response
  if (rsp.ContentEncoding.Equals("deflate"))
    stream = new DeflateStream(stream, CompressionMode.Decompress);
```

[6] The section on parsing responses and displaying the run-time output is identical to that of an HTTP GET request. Therefore, these sections simply refer to those in HTTP GET response processing.

```
  else if (rsp.ContentEncoding.Equals("gzip"))
    stream = new GZipStream(stream, CompressionMode.Decompress);
  .
  .
  .
```

Parsing the XML response

The parsing code is the same as for an HTTP GET request (See section 2.3.1 Using C# (.NET 2.0) Parsing the XML Response).

Program output

The sample output is the same as for an HTTP GET request (See section 2.3.1 Using C# (.NET 2.0)Program Output).

2.5.2 Using Java

Issuing an HTTP POST request

This example uses Java to make a POST request and parse the response.

❖ **Since there is a java.net.ContentHandler and a org.xml.sax.ContentHandler, applications should avoid the following code:**
import java.net.*;
import org.xml.sax.*;

```
import java.io.*;
import java.util.zip.*;
import java.net.URL;
import javax.net.ssl.HttpsURLConnection;
import javax.xml.parsers.*;
import org.xml.sax.*;

public class MyXMLParser implements ContentHandler {

public static void main(String[] args)
{
  try {
    // setup request
    URL url = new URL(
          "https://datadirect-beta.factset.com/services/DFSnapshot");
    HttpsURLConnection conn = (HttpsURLConnection)url.openConnection();
    conn.setDoOutput(true);
    conn.setRequestProperty("Accept-Encoding", "deflate, gzip");
    conn.setRequestProperty("Authorization",
                    "Basic AaBbCcDdEeFfGgHhIi1234==");
    conn.setRequestMethod("POST");

    // Fill in the Request Stream with the POST data
    String postData = "serv=FDS_C&req_id=99&format=xml" +
          "&ids=FDS-USA,IBM-USA,TWX-USA&fields=ASK_1,BID_1,LAST_1";
    OutputStream wr = conn.getOutputStream();
    wr.write(postData.getBytes());
    wr.flush();

    // getInputStream will implicitly connect and get the response
    InputStream inputStream = conn.getInputStream();

                                                  // uncompress the response
    if (conn.getContentEncoding().equals("deflate"))
      inputStream = new InflaterInputStream(inputStream,
                                  new Inflater(true));
    else if (conn.getContentEncoding().equals("gzip"))
      inputStream = new GZIPInputStream(inputStream);
    .
    .
```

Parsing the XML response

The parsing code is the same as for an HTTP GET request (See section 2.3.2 Using Java Parsing the XML Response).

Program output

The sample output is the same as for an HTTP GET request (See section 2.3.2 Using Java Program Output).

Chapter 3 Processing Responses

3.1 Response when Specifying Fields Explicitly

3.1.1 XML Response with Specified Fields

This example requests the ASK_1, BID_1, and LAST_1 fields for the symbols, FDS-USA, IBM-USA, and TWX-USA, on the FDS_C data service.

https://datadirect-beta.factset.com/services/DFSnapshot?serv=FDS_C&req_id=99&format=xml&ids=FDS-USA,IBM-USA,TWX-USA&fields=ASK_1,BID_1,LAST_1

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
<Response Service="FDS_C">
 <Request id="99">
  <RequestedSymbols>FDS-USA,IBM-USA,TWX-USA</RequestedSymbols>
  <RequestedFields>ASK_1,BID_1,LAST_1</RequestedFields>
  <Host>dfstagea01[3978]</Host>
 </Request>
 <Error code="0" description="" />
 <Records>
  <Record req_sym="FDS-USA" key="FDS-USA" stale="">
   <Fields>
    <Field id="200" name="ASK_1" value="43.75" />
    <Field id="100" name="BID_1" value="43.73" />
    <Field id="300" name="LAST_1" value="43.73" />
   </Fields>
  </Record>
  <Record req_sym="IBM-USA" key="IBM-USA" stale="">
   <Fields>
    <Field id="200" name="ASK_1" value="76.18" />
    <Field id="100" name="BID_1" value="76.17" />
    <Field id="300" name="LAST_1" value="76.19" />
   </Fields>
  </Record>
  <Record req_sym="TWX-USA" key="TWX-USA" stale="">
   <Fields>
    <Field id="200" name="ASK_1" value="16.34" />
    <Field id="100" name="BID_1" value="16.33" />
    <Field id="300" name="LAST_1" value="16.33" />
   </Fields>
  </Record>
 </Records>
</Response>
```

3.1.2 CSV Response with Specified Fields

If the response format specifies ”csv”, a CSV file will be sent.

https://datadirect-beta.factset.com/services/DFSnapshot?serv=FDS_C&req_id=99&format=csv&ids=FDS-USA,IBM-USA,TWX-USA&fields=ASK_1,BID_1,LAST_1

The URL above produces the following CSV response:

```
REQ_SYM,KEY,ASK_1,BID_1,LAST_1
FDS-USA,FDS-USA,53.80,53.77,53.77
IBM-USA,IBM-USA,93.78,93.77,93.78
TWX-USA,TWX-USA,20.46,20.45,20.46
```

3.1.3 JSON Response with Specified Fields

This request is for the same data as the previous example but returns the response in JSON format.

https://datadirect-beta.factset.com/services/DFSnapshot?serv=FDS_C&req_id=99&format=json&ids=FDS-USA,IBM-USA,TWX-USA&fields=ASK_1,BID_1,LAST_1


```
{
	"Error": [0,
	""],
	"FDS-USA": {
		"Fields": {
			"100": {
				"BID_1": "112.8200"
			},
			"200": {
				"ASK_1": "114.1100"
			},
			"300": {
				"LAST_1": "114.0800"
			}
		},
		"Key": "FDS-USA",
		"Stale": ""
	},
	"Host": "DFSTAGEA06[1011062]",
	"IBM-USA": {
		"Fields": {
			"100": {
				"BID_1": "179.84"
			},
			"200": {
				"ASK_1": "179.8500"
			},
			"300": {
				"LAST_1": "179.84"
			}
		},
		"Key": "IBM-USA",
		"Stale": ""
	},
	"Request ID": "99",
	"Requested Fields": "ASK_1,BID_1,LAST_1",
	"Requested Symbols": "FDS-USA,IBM-USA,TWX-USA",
	"Service": "FDS_C",
	"TWX-USA": {
		"Fields": {
			"100": {
				"BID_1": "65.9000"
			},
			"200": {
				"ASK_1": "65.95"
			},
			"300": {
				"LAST_1": "65.94"
			}
		},
		"Key": "TWX-USA",
		"Stale": ""
	}
}
```


3.1.4 XML Response with All Fields

This example contains all the fields for the symbol FDS-USA (since the fields variable is not present).

https://datadirect-beta.factset.com/services/DFSnapshot?serv=FDS_C&req_id=99&format=xml&ids=FDS-USA

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
<Response Service="FDS_C">
 <Request id="99">
 <RequestedSymbols>FDS-USA</RequestedSymbols>
 <RequestedFields />
 <Host>dfstagea01[4034]</Host>
</Request>
<Error code="0" description="" />
<Records>
 <Record req_sym="FDS-USA" key="FDS-USA" stale="">
  <Fields>
   <Field id="100" name="BID_1" value="53.36" />
   <Field id="102" name="BID_TIME_1" value="105558000" />
   <Field id="104" name="BID_VOL_1" value="1" />
   <Field id="106" name="BID_TICK_1" value="1" />
   <Field id="107" name="BID_EXCH_1" value="1" />
   <Field id="200" name="ASK_1" value="53.36" />
   <Field id="202" name="ASK_TIME_1" value="105558000" />
   <Field id="204" name="ASK_VOL_1" value="1" />
   <Field id="207" name="ASK_EXCH_1" value="6" />
   <Field id="293" name="QUOTE_COND" value="0" />
   <Field id="300" name="LAST_1" value="53.33" />
   <Field id="302" name="LAST_TIME_1" value="105455000" />
   <Field id="304" name="LAST_VOL_1" value="100" />
   <Field id="305" name="LAST_COND_1" value="0" />
   <Field id="306" name="LAST_TICK_1" value="3" />
   <Field id="307" name="LAST_EXCH_1" value="1" />
   <Field id="601" name="CUM_VOL" value="23500" />
   <Field id="603" name="VWAP" value="53.315628" />
   <Field id="604" name="TRD_CNT" value="118" />
   <Field id="605" name="BLK_TRD_CNT" value="0" />
   <Field id="606" name="BLK_CUM_VOL" value="0" />
   <Field id="610" name="PREV_CLOSE" value="53.22" />
   <Field id="710" name="OPEN_1" value="53.17" />
   <Field id="720" name="HIGH_1" value="53.44" />
   <Field id="723" name="LOW_1" value="53.15" />
   <Field id="2027" name="CUSIP" value="30307510" />
   <Field id="2034" name="SECURITY_STATUS" value="0" />
  </Fields>
 </Record>
</Records>
</Response>
```

3.1.5 CSV Response with All Fields

This example contains all the fields for the symbol FDS-USA in a CSV response (since the fields variable is not present).

https://datadirect-beta.factset.com/services/DFSnapshot?serv=FDS_C&req_id=99&format=csv&ids=FDS-USA

The URL above produces the following CSV response:

```
REQ_SYM,KEY,LAST_1,BID_1,BID_TIME_1,BID_VOL_1,BID_TICK_1,BID_EXCH_1,ASK_1,ASK_TIME_1,ASK_VOL_1,ASK_EXCH_1,QUOTE_COND,LAST_TIME_1,LAST_VOL_1,LAST_COND_1,LAST_TICK_1,LAST_EXCH_1,CUM_VOL,VWAP,TRD_CNT,BLK_TRD_CNT,BLK_CUM_VOL,PREV_CLOSE,OPEN_1,HIGH_1,LOW_1,CUSIP,SECURITY_STATUS
FDS-USA,FDS-USA,54.19,121318000,2,4,1,54.21,121318000,3,1,0,121316000,100,0,3,1,40200,54.235388,255,0,0,54.11,54.19,54.35,54.06,30307510,0
```

- If the symbols specified in the CSV do not contain the same fields (i.e., are of different security types), multiple CSV files are returned. Each file is separated by a new-line character, and each file will have its own header row describing each field. This only happens if the request does not specify the fields explicitly. To make sure that a single CSV is ALWAYS returned, applications should ALWAYS specify the fields parameter when requesting CSV files (see section *3.1.7 Forcing a Matrix Response*).

3.1.6 JSON Response with All Fields (format = 'json')

This example contains all the fields for the symbol FDS-USA in a JSON response (since the fields variable is not present).

https://datadirect-beta.factset.com/services/DFSnapshot?serv=FDS_C&req_id=99&format=json&ids=FDS-USA


```
{
  "Error": [
    0,
    ""
  ],
  "FDS-USA": {
    "Fields": {
      "5": {
        "EXCHANGE": "11099"
      },
      "13": {
        "ORIG_SEQUENCE": ""
      },
      "100": {
        "BID_1": "113.3300"
      },
      "101": {
        "BID_DATE_1": "20131121"
      },
      "102": {
        "BID_TIME_1": "104058267"
        "PRICE_CURRENCY": "USD"
      },
       .
       .
       .
       .
      "2034": {
        "SECURITY_STATUS": "0"
      },
      "2037": {
        "GMT_OFFSET": "-300"
      },
      "2042": {
        "FINANCIAL_STATUS": "4"
      },
      "2045": {
        "HALT_INFO": "0"
      },
```

```
      "2573": {
        "AVG_30DAY_VOL": "0.2944"
      },
      "2574": {
        "AVG_5DAY_VOL": "0.2429"
      },
      "2709": {
        "TRADE_CONDITION": "00 0"
      }
    },
    "Key": "FDS-USA",
    "Stale": ""
  },
  "Host": "MDSSTAGEA03[48]",
  "Request ID": "99",
  "Requested Fields": "",
  "Requested Symbols": "FDS-USA",
  "Service": "FDS_C"
}
```

3.1.7 Forcing a Matrix Response

When the request specifies the fields of interest, the response will be forced into a matrix or table. Invalid records and fields, in addition to records which are not accessible based on the user's permissions, will return special field values. These values are listed in section 3.3 Special Field Values. Additionally, the sorting order of the records and fields will be identical to that in the request. This holds true for both CSV and XML response formats.

This request is for data that does not exist, as well as data that is not permissioned.

https://datadirect-beta.factset.com/services/DFSnapshot?serv=FDS_C&req_id=99&format=xml&ids=FDS-USA,XX-USA,US10Y-TU1&fields=ASK_1,BID_1,LAST_1

XML Output:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
<Response Service="FDS_C">
 <Request id="99">
  <RequestedSymbols>FDS-USA,XX-USA,US10Y-TU1</RequestedSymbols>
  <RequestedFields>ASK_1,BID_1,LAST_1</RequestedFields>
  <Host>dfstagea01[4108]</Host>
 </Request>
 <Error code="0" description="" />
 <Records>
  <Record req_sym="FDS-USA" key="FDS-USA" stale="">
   <Fields>
    <Field id="200" name="ASK_1" value="43.67" />
    <Field id="100" name="BID_1" value="43.64" />
    <Field id="300" name="LAST_1" value="43.65" />
   </Fields>
  </Record>
  <Record req_sym="XX-USA" key="#NF#" stale="Symbol not found">
   <Fields>
    <Field id="200" name="ASK_1" value="#NF#" />
    <Field id="100" name="BID_1" value="#NF#" />
    <Field id="300" name="LAST_1" value="#NF#" />
   </Fields>
  </Record>
  <Record req_sym="US10Y-TU1" key="#NE#" stale="Not Entitled {158}">
   <Fields>
    <Field id="200" name="ASK_1" value="#NE#" />
    <Field id="100" name="BID_1" value="#NE#" />
    <Field id="300" name="LAST_1" value="#NE#" />
   </Fields>
  </Record>
 </Records>
</Response>
```

CSV Output (change format in request to 'csv'):

```
REQ_SYM,KEY,ASK_1,BID_1,LAST_1
FDS-USA,FDS-USA,53.80,53.77,53.77
XX-USA,#NF#,#NF#,#NF#
US10Y-TU1,#NE#,#NE#,#NE#
```

3.1.8 XML Response Header

Each XML response will have a response header. The header contains the request id, requested symbols, and requested fields. In addition, if a request is invalid an error code along with a description will be sent via the header.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
<Response Service="FDS_C">
 <Request id="99">
  <RequestedSymbols>FDS-USA</RequestedSymbols>
  <RequestedFields />
  <Host>dfstagea01[4162]</Host>
 </Request>
 <Error code="0" description="" />
 .
 .
 .
```

3.1.9 XML Response Body

The response body contains the data for all items requested. For each data item, a record will be generated in the output stream. Records contain multiple fields and each field will have a name, id (the field identifier), and value.

```
.
 .
 .
 <Records>
  <Record req_sym="FDS-USA" key="FDS-USA" stale="">
   <Fields>
    <Field id="100" name="BID_1" value="53.70" />
    <Field id="102" name="BID_TIME_1" value="143946000" />
    <Field id="104" name="BID_VOL_1" value="5" />
                                                    .
                                                    .
                                                    .
    <Field id="605" name="BLK_TRD_CNT" value="1" />
    <Field id="710" name="OPEN_1" value="53.17" />
    <Field id="720" name="HIGH_1" value="53.8592" />
    <Field id="723" name="LOW_1" value="53.15" />
    <Field id="2027" name="CUSIP" value="30307510" />
    <Field id="2034" name="SECURITY_STATUS" value="0" />
   </Fields>
  </Record>
 </Records>
</Response>
```

3.2 Errors in the Response

The Exchange Snapshot Service does not send HTTP errors. Instead, all errors are included in the error element or field values. For XML responses, errors will be contained in the XML header (See section 3.1.8 XML Response Header). CSV responses contain the error in the CSV data (since there is no header).

Error codes are sent in the XML response header or CSV file. The following values are possible:

Error Code	Meaning
0	No Error
402	Invalid HTTP method. Either the method is not POST or GET or exceeds the maximum request length (currently set at 5000 bytes). The description field will indicate the exact reason.
403	Invalid HTTP URL. Necessary query parameters are missing in the request. The description field will indicate the exact reason.
404	Access Denied. The user, password, or both is invalid. Contact FactSet Consulting Services for assistance.

3.2.1 Invalid Method

If a request is not a POST/GET or is larger than 5000 bytes, the following error is generated:

XML Output:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
<Response Service="FDS_C">
  <Request id="99">
    <RequestedSymbols />
    <RequestedFields />
    <Host>dfstagea01[4281]</Host>
  </Request>
  <Error code="402" description="Header too large" />
</Response>
```

CSV Output:

```
Service,id,Requested Symbols,Requested Fields,Host,Error Code,Error Description
FDS_C,99,,,dfstagea01[4352],402,Header too large
```

3.2.2 Invalid URL

If there are no symbols in the query string, the following error is generated:

XML Output:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
<Response Service="FDS_C">
  <Request id="99">
    <RequestedSymbols />
    <RequestedFields>ASK_1,BID_1,LAST_1</RequestedFields>
    <Host>dfstagea01[4439]</Host>
  </Request>
  <Error code="403" description="No symbols or chain specified in the Query
                                 String" />
</Response>
```

CSV Output:

```
Service,id,Requested Symbols,Requested Fields,Host,Error Code,Error Description
FDS_C,99,FDS-USA,"ASK_1,BID_1,LAST_1",dfstagea01[4508],403,No symbols or chain specified in the Query String
```

3.2.3 Access Denied

If a username or password is invalid, the following error is generated.

XML Output:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
<Response Service="FDS_C">
  <Request id="99">
    <RequestedSymbols>FDS-USA</RequestedSymbols>
    <RequestedFields>ASK_1,BID_1,LAST_1</RequestedFields>
    <Host>dfstagea01[4662]</Host>
  </Request>
  <Error code="404" description="Access Denied" />
</Response>
```

CSV Output:

```
Service,id,Requested Symbols,Requested Fields,Host,Error Code,Error Description
FDS_C,99,FDS-USA,"ASK_1,BID_1,LAST_1",dfstagea01[4719],404,Access Denied
```

3.3 Special Field Values

Special values may be sent to indicate certain types of error conditions. These values are used when attempting to force the response into a matrix for both CSV and XML formats.

Special Field Value	Meaning
#NF#	Not Found. The requested symbol could not be found by the service.
#NA#	Not Applicable. The requested field could not be found for the requested symbol. This usually means that the field is not applicable for this type of record,
#NE#	Not Entitled. The supplied username and password is not entitled to the data requested.
#NS#	No Service. The requested service is not available. Check to make sure the service name is spelled correctly.

3.3.1 Symbol Not Found

This request generates a valid response for the first symbol, but the second symbol is invalid.

https://datadirect-beta.factset.com/services/DFSnapshot?serv=FDS_C&req_id=99&format=xml&ids=FDS-USA,XX-USA&fields=ASK_1,BID_1,LAST_1

XML Output:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
<Response Service="FDS_C">
 <Request id="99">
  <RequestedSymbols>FDS-USA,XX-USA</RequestedSymbols>
  <RequestedFields>ASK_1,BID_1,LAST_1</RequestedFields>
  <Host>dfstagea01[4759]</Host>
 </Request>
 <Error code="0" description="" />
 <Records>
  <Record req_sym="FDS-USA" key="FDS-USA" stale="">
   <Fields>
    <Field id="200" name="ASK_1" value="43.67" />
    <Field id="100" name="BID_1" value="43.64" />
    <Field id="300" name="LAST_1" value="43.65" />
   </Fields>
  </Record>
  <Record req_sym="XX-USA" key="#NF#" stale="Symbol not found">
   <Fields>
    <Field id="200" name="ASK_1" value="#NF#" />
    <Field id="100" name="BID_1" value="#NF#" />
    <Field id="300" name="LAST_1" value="#NF#" />
   </Fields>
  </Record>
 </Records>
</Response>
```

CSV Output:

```
REQ_SYM,KEY,ASK_1,BID_1,LAST_1
FDS-USA,FDS-USA,54.26,54.23,54.24
XX-USA,#NF#,#NF#,#NF#,#NF#
```

3.3.2 Field Not Applicable

Since the LAST_1 field does not apply to a FOREX Record, this request generates a #NA# value in the AUD-TU1 (Australian dollar) record.

https://datadirect-beta.factset.com/services/DFSnapshot?serv=FDS_C&req_id=99&format=xml&ids=FDS-USA,AUD-TU1&fields=ASK_1,BID_1,LAST_1

XML Output:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
<Response Service="FDS_C">
 <Request id="99">
  <RequestedSymbols>FDS-USA,AUD-TU1</RequestedSymbols>
  <RequestedFields>ASK_1,BID_1,LAST_1</RequestedFields>
  <Host>dfstagea01[4836]</Host>
 </Request>
 <Error code="0" description="" />
 <Records>
  <Record req_sym="FDS-USA" key="FDS-USA" stale="">
   <Fields>
    <Field id="200" name="ASK_1" value="43.67" />
    <Field id="100" name="BID_1" value="43.64" />
    <Field id="300" name="LAST_1" value="43.67" />
   </Fields>
  </Record>
  <Record req_sym="AUD-TU1" key="AUD-TU1" stale="">
   <Fields>
    <Field id="200" name="ASK_1" value="0.7636" />
    <Field id="100" name="BID_1" value="0.7631" />
    <Field id="300" name="LAST_1" value="#NA#" />
   </Fields>
  </Record>
 </Records>
</Response>
```

CSV Output:

```
REQ_SYM,KEY,ASK_1,BID_1,LAST_1
FDS-USA,FDS-USA,54.26,54.23,54.24
AUD-TU1,AUD-TU1,0.7636,0.7631,#NA#
```

3.3.3 Permission Errors (Not Entitled)

https://datadirect-beta.factset.com/services/DFSnapshot?serv=FDS_C&req_id=99&format=xml&ids=US10Y-TU1&fields=ASK_1,BID_1,LAST_1

XML Output:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
<Response Service="FDS_C">
  <Request id="99">
    <RequestedSymbols>US10Y-TU1</RequestedSymbols>
    <RequestedFields>ASK_1,BID_1,LAST_1</RequestedFields>
    <Host>dfstagea01[4910]</Host>
  </Request>
  <Error code="0" description="" />
  <Records>
    <Record req_sym="US10Y-TU1" key="#NE#" stale="Not Entitled {158}">
      <Fields>
        <Field id="200" name="ASK_1" value="#NE#" />
        <Field id="100" name="BID_1" value="#NE#" />
        <Field id="300" name="LAST_1" value="#NE#" />
      </Fields>
    </Record>
  </Records>
</Response>
```

CSV Output:

```
REQ_SYM,KEY,ASK_1,BID_1,LAST_1
US10Y-TU1,#NE#,#NE#,#NE#,#NE#
```

3.3.4 Connection or Service Not Available

https://datadirect-beta.factset.com/services/DFSnapshot?serv=US&req_id=99&format=xml&ids=FDS-USA&fields=ASK_1,BID_1,LAST_1

XML Output:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
<Response Service="US">
  <Request id="99">
    <RequestedSymbols>FDS-USA</RequestedSymbols>
    <RequestedFields>ASK_1,BID_1,LAST_1</RequestedFields>
    <Host>dfstagea01[5081]</Host>
  </Request>
  <Error code="0" description="" />
  <Records>
    <Record req_sym="FDS-USA" key="#NS#" stale="Service not connected">
      <Fields>
        <Field id="200" name="ASK_1" value="#NS#" />
        <Field id="100" name="BID_1" value="#NS#" />
        <Field id="300" name="LAST_1" value="#NS#" />
      </Fields>
    </Record>
  </Records>
</Response>
```

CSV Output:

```
REQ_SYM,KEY,ASK_1,BID_1,LAST_1
FDS-USA,#NS#,#NS#,#NS#,#NS#
```

Chapter 4.0 Portfolios

4.1 Introduction

You can store your portfolios in FactSet. There is a standard file type, OFDB, that most portfolios are stored in and some standard fields, such as SHARES and COST. The Snapshot Service has the ability to retrieve those portfolios in a secure way, and the make subsequent request for each individual security in the portfolio. This ability saves you from having to manage your own ticker lists. Please contact your FactSet representative if you wish to learn more about portfolio storage with FactSet.

4.2 Resolving Portfolios Using the Snapshot Service

Portfolio requests through the Exchange Snapshot Service use the query string parameter 'chain'. A chain request makes two sets of requests. The first request retrieves a list of symbols and the second makes requests for that list. The service acts as if it has been passed a large list of symbols in the 'ids' query parameter. Using both 'ids' and 'chain' in the same request will combine the two lists of symbols together.

4.3 FactSet Partner Keys HTTP Authentication

If you make portfolio requests, FactSet requires you to connect to FactSet OnDemand using FactSet Partner Keys HTTP Authentication. This scheme is stronger than the basic authentication normally used. It was developed by FactSet and involves rotating keys on a regular basis. More details on this authentication scheme and how to implement it can be found in FactSet's document *FactSet Partner Keys*.

4.4 Portfolio Request Key Format

When requesting a portfolio, the key follows the following format:

[Portfolio Name]-FCHAIN:p

- "-FCHAIN" identifies this key as a FactSet chain. Anything, including multiple dashes, is allowed to the left of this string.
- ":p" identifies this FactSet chain as a portfolio.
- The *Portfolio Name* follows the common FactSet file standards:

  - Location:Name.Extension
  - Common values of *Location* are FACTSET: (common to all clients), CLIENT: (common to all serial numbers of a username), and PERSONAL: (specific to the requesting serial number).
  - *Name* can be anything.
  - Common values of *Extension*, which identifies the type of portfolio, are OFDB, PRT, and CSTM.

Here is an example portfolio key:

FACTSET:DOWJONES.OFDB-FCHAIN:p

This will request FactSet's Dow Jones Industrial Average constituents' portfolio. It is available to all clients and is a good test portfolio.

4.5 Portfolio Example

The following request returns the ASK_1, BID_1, and LAST_1 fields for the symbol IBM-USA and portfolio FACTSET:DOWJONES.OFDB on the FDS_C data service:

https://datadirect-beta.factset.com/services/DFSnapshot?serv=FDS_C&req_id=99&format=xml&ids=IBM-USA&chain=FACTSET:DOWJONES.OFDB-FCHAIN:p&fields=ASK_1,BID_1,LAST_1

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
<Response Service="FDS1">
 <Request id="99">
  <RequestedChain>FACTSET:DOWJONES.OFDB-FCHAIN:P</RequestedChain>
  <RequestedFields />
  <Host>dfstagea01[153405]</Host>
    </Request>
 <Error code="0" description="" />
 <Records>
  <Record req_sym="IBM-USA" key="IBM-USA" stale="">
   <Fields>
    <Field id="200" name="ASK_1" value="80.09" />
    <Field id="100" name="BID_1" value="80.07" />
    <Field id="300" name="LAST_1" value="80.0856" />
      </Fields>
     </Record>
 <Records>
  <Record req_sym="69331C10" key="PCG-USA" stale="">
   <Fields>
    <Field id="200" name="ASK_1" value="37.45" />
    <Field id="100" name="BID_1" value="37.4400" />
    <Field id="300" name="LAST_1" value="37.4400" />
      </Fields>
     </Record>
  .
  .
  .
  <Record req_sym="50075N10" key="KFT-USA" stale="">
   <Fields>
    <Field id="200" name="ASK_1" value="26.90" />
    <Field id="100" name="BID_1" value="26.8900" />
    <Field id="300" name="LAST_1" value="26.89" />
      </Fields>
     </Record>
   </Records>
  </Response>
```

Chapter 5.0 FactSet Data Link

5.1 Introduction

FactSet Data Link allows individual users to make requests using their terminal permissions. A FactSet Workstation with access to FactSet Data Link must be installed for this service to work, and the user must be logged into FactSet.

Users of the Exchange Snapshot Service will, in most cases, need exchange redistribution agreements in order to share the data internally or externally. The client is also responsible for managing their permissions. With FactSet Data Link, the exchange redistribution agreements are not necessary as users are not allowed to share or distribute market data information outside of the FactSet terminal. In addition, FactSet maintains the individual users' permission maps. The FactSet Data Link response will be in the same format as the Exchange Snapshot Service.

The individual will authenticate for the service the first time they log in or if there are any changes to the workstation setup.

5.2 How FactSet Data Link works

Each individual user needs to be permissioned by FactSet in order to use the FactSet Data Link. Before making a request, the client application must use FactSet Workstation to authenticate the individual user

5.3 Differences between FactSet Data Link and the Enterprise Service

The responses between FactSet Data Link and the Enterprise Snapshot are exactly the same; however, FactSet Data Link requests require some additional steps to be taken.

Instead of making an https request directly to the Snapshot service the request is made using COM. Please see the below example for how the COM call is made.

Calling this function will start FactSet and prompt a login if necessary. The RunApplication function will return ERROR_SUCCESS when the service was successfully hit, or appropriate COM errors for invalid usage.

5.4 Generic C# Example

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Reflection; // This is needed for BindingFlags
using System.IO;
using System.IO.Compression;
namespace DatalinkSnapshotLatest
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                Type fds_obj_type = Type.GetTypeFromProgID("FactSet.FactSet_API");
                object fds_obj = Activator.CreateInstance(fds_obj_type);
                string[] runapp_params = new string[] { "DataDirect", "{\"serviceName\" :
\"DFSnapshot\", \"timeoutMS\" : 30000, \"method\" : \"GET\", \"parameters\" : { \"id\" : \"VOD-
LON\", \"fields\" : \"LAST_1,LAST_VOL_1,LAST_TIME_1\"}}" };

                System.Byte[] ret = (System.Byte[])fds_obj_type.InvokeMember("RunApplication",
BindingFlags.InvokeMethod, null, fds_obj, runapp_params);

                var str = System.Text.Encoding.UTF8.GetString(ret);
                writecontents(str);
            }
            catch (Exception e)
            {
                writecontents(e.Message);
            }
        }
        static void writecontents(string a)
        {
            string path = @"C:\ProgramData\LogFilefds100.txt";
            try
            {
                // This text is added only once to the file.
                if (!File.Exists(path))
                {
                    // Create a file to write to.
                    using (StreamWriter sw = File.CreateText(path))
                    {
                    }
                }
                using (StreamWriter sw = File.AppendText(path))
                {
                    sw.WriteLine(a);
                }
            }
            catch (Exception e)
            {
                String d = e.Message;
                writecontents(d);
            }
        }
    }
}
```

5.5 Java example

Example that uses Jacob and Java's JSON reference implementation.

```
package datalink_ex;

import com.jacob.com.Dispatch;

class DFSnapshot_Example extends com.jacob.com.Dispatch {
	public DFSnapshot_Example() {
		super("FactSet.FactSet_API");
		Dispatch.call(this,  "GetOnline");
	}

	public String request1() {
		String dfsnapshot_ex = new org.json.JSONObject()
				.put("serviceName", "DFSnapshot")
				.put("timeoutMS",  30000)
				.put("method", "GET")
				.put("parameters", new org.json.JSONObject()
						.put("id", "VOD-LON")
						.put("fields", "LAST_1,LAST_VOL_1,LAST_TIME_1")
				).toString();

		return Dispatch.call(this, "RunApplication", "DataDirect", dfsnapshot_ex).toSa
feArray().asString();
	}
}

public class Main {
	public static void main(String[] args) {
		System.out.println(new DFSnapshot_Example().request1());
	}
}
```

Appendix A: Using C#'s XMLDocument

This example code illustrates parsing an XML Response using C#'s XMLDocument. It loads the XML into an XMLDocument instance and then uses XPath to select nodes of the XMLDocument and print out the relevant data.

```
                                                                    .
 .
 // Get XML Response using GET or POST (see 2.3 or 2.4)
 .
 .
 XmlTextReader reader = new XmlTextReader(stream);
 reader.WhitespaceHandling = WhitespaceHandling.None;

 XmlDocument doc = new XmlDocument();
 doc.Load(reader);

 foreach (XmlNode node in doc.SelectNodes("/Response/Error"))
 {
   System.Console.WriteLine(node.Name);
   printAttributes(node);
   System.Console.WriteLine();
 }

 foreach (XmlNode node in doc.SelectNodes("/Response/Records/Record"))
 {
   System.Console.WriteLine(node.Name);
   printAttributes(node);

   foreach (XmlNode fieldNode in node.SelectNodes("Fields/Field"))
     printAttributes(fieldNode);

   System.Console.WriteLine();
 }
}

static void printAttributes(XmlNode node)
{
 foreach (XmlAttribute att in node.Attributes)
   System.Console.Write(att.Name + ": " + att.Value + ", ");
 System.Console.WriteLine();
}

} // class MyXMLParser
```

Appendix B: Using Java's Document Class

This example code illustrates parsing an XML Response using Java's Document Class. It loads the XML into a Document instance and then uses XPath to select nodes of the Document and print out the relevant data[7].

```
import java.io.*;
import java.util.zip.*;
import java.net.URL;
import javax.net.ssl.HttpsURLConnection;
import javax.xml.parsers.*;
import javax.xml.xpath.*;
import org.w3c.dom.*;
import org.xml.sax.*;

public class MyDOMXPathParser {

public static void main(String[] args)
{
  try {

    // Get XML Response using GET or POST (see 2.3 or 3.4)

    // Parse the XML into a DOM Document
    DocumentBuilderFactory docFactory =
       DocumentBuilderFactory.newInstance();
    docFactory.setNamespaceAware(true);
    DocumentBuilder docBuilder = docFactory.newDocumentBuilder();
    Document doc = docBuilder.parse(inputStream);

    // Set up an XPath for easy access of XML structure
    XPath xp = XPathFactory.newInstance().newXPath();

    // Go through the XML Response and print out the relevant data
    NodeList nodes = (NodeList) xp.evaluate("/Response/Error",
       doc.getDocumentElement(), XPathConstants.NODESET);
    for (int i = 0; i < nodes.getLength(); ++i)
    {
      Node errorNode = nodes.item(i);
      System.out.println(errorNode.getNodeName());
      printAttributes(errorNode);
      System.out.println();
    }

    nodes = (NodeList) xp.evaluate("/Response/Records/Record",
       doc.getDocumentElement(), XPathConstants.NODESET);
    for (int i = 0; i < nodes.getLength(); ++i)
    {
      Node recordNode = nodes.item(i);
      System.out.println(recordNode.getNodeName());
      printAttributes(recordNode);

      NodeList fieldNodes = (NodeList) xp.evaluate("Fields/Field",
         recordNode, XPathConstants.NODESET);


      for (int k = 0; k < fieldNodes.getLength(); ++k)
      {
        Node fieldNode = fieldNodes.item(k);
        printAttributes(fieldNode);
      }
      System.out.println();
    }
  } catch (IOException e) {
    // Deal with IOException
```

[7] This method of parsing does not use event-driven callbacks, and thus the structure is modified accordingly.

```
  } catch (XPathExpressionException e) {
    // Deal with XPathExpressionException
  } catch (ParserConfigurationException e) {
    // Deal with ParserConfigurationException
  } catch (SAXException e) {
    // Deal with SAXException
  }
}

// Helper function that prints out all attributes of a node
private static void printAttributes(Node node)
{
  NamedNodeMap atts = node.getAttributes();
  for (int i = 0; i < atts.getLength(); ++i)
  {
    Node att = atts.item(i);
    System.out.print(att.getNodeName() + ": " + att.getNodeValue()
        + ", ");
  }
  System.out.println();
}

} // class MyXMLParser
```

Appendix C: Document Revisions

The following are revisions made since *FactSet Exchange DataFeed Snapshot Service version 1.1.l*.

Revision(s)	Section(s)
Added explicit security limit, 500.	2.2
New chapters.	5
New chapter.	6
Changed env=dev to env=pub	6.5
Changed DataDirect references to FactSet OnDemand	All
Changed pub queries to use DFSnapshotStaging	All

The following are revisions made since *FactSet Exchange DataFeed Snapshot Service version 1.1s*

Revision(s)	Section(s)
Changed Staging queries to use beta	All
Added JSON examples	4.1
Combined GET and POST chapter	2

The following are revisions made since *FactSet Exchange DataFeed Snapshot Service version 2.0*

Revision(s)	Section(s)
Updated Data Link request	5